

WebSocket protokol

WebSockets predstavljaju tehnologiju koja omogućava otvaranje interaktivne komunikativne sesije između klijenta i servera. Predstavlja se i kao nadogradnja za HTTP protokol. Na ovaj način korisnik (klijent) može slati poruke serveru i dobijati odgovore vođene događajima (event-driven programming), bez potrebe da klijent konstantno šalje zahteve serveru sa porukom *da li je pristiglo nešto novo* (dugačko anketiranje, engl. long polling). Ovakvih scenarija ima dosta, a najlakše se primećuju u čet aplikacijama kada je veoma bitno da se u realnom vremenu dobijaju i šalju nove poruke.

HTTP dugačko anketiranje

Istorijski, gledano pre WebSoketa se koristio princip HTTP dugačkog anketiranja (HTTP long polling). Ovom metodom klijent je konstantno slao zahteve serveru sa pitanjem *da li postoji nova informacija za mene*. Server bi održavao taj zahtev otvorenim dokle god se taj neki novi podatak za klijenta ne bi pojavio (ili dok ne dođe do *request timed out* (408) greške koja označava da je zahtev istekao). Suština je da klijent konstantno ispituje (polluje) server o podatku da li je prispeo novi deo informacije za njega. Kada bi klijent konačno dobio odgovor na taj zahtev, odmah bi slao sledeći zahtev i čekao. Ovakav proces bi se konstantno ponavljao, što nas dovodi i do mana ovakvog pristupa.

Sa rastom baze podataka na serveru i broja novih klijenata, ovakav pristup (HTTP long polling) zahteva sve veće iskorišćenje CPU-a, protoka interneta kao i skladištenja. Ovo vrlo brzo postaje jako komplikovan sistem sa previše podataka i troškova. Takođe raste i broj nepotrebnih zahteva, jer se za svaki zahtev mora ostvariti nova konekcija.

Napomena

Pored dugačkog anketiranja gde server, nakon primljenog zahteva, ne šalje odgovor dok ne dobije tražene podatke (ili dok ne istekne trajanje zahteva), postoji i kratko anketiranje (short polling). Kratko anketiranje je kada klijent šalje probni signal periodično, na svakih x milisekundi, što znači da ćemo svakih x sekundi dobijati podatak. Ovaj pristup nije efikasan jer *zatrpava* server zahtevima po kojima će dobijeni odgovori u većini slučajeva biti prazni.

WebSockets

WebSockets nam omogućavaju ostvarivanje jedne TCP konekcije između klijenta i servera koja omogućava dvostranu komunikaciju (full-duplex). Zahtev za WebSocket konekciji se šalje serveru sa klijentske strane kroz proces poznat kao WebSocket usaglašavanje. Ovaj zahtev počinje kao običan HTTP zahtev ka serveru i njegov deo obuhvata i Upgrade polje zaglavlja, koje nagoveštava serveru kako klijent pokušava da napravi WebSocket konekciju.

Primer zaglavlja zahteva

GET /index.html HTTP/1.1

Upgrade: WebSocket

Connection: Upgrade

Primer zaglavlja odgovora

HTTP/1.1. 101 WebSocket Protocol Handshake

Upgrade: WebSocket

Connection: Upgrade

Ako server podržava WebSocket protokol i prihvati zahtev, inicijalna HTTP konekcija se menja u WebSocket konekciju koristeći isti, TCP protokol, i održava se kroz čitavu sesiju. Ovakva konekcija nam omogućava da se podaci u realnom vremenu šalju ka klijentu po potrebi. Na taj način eliminišemo potrebu za dugačkim anketiranjem servera, smanjujemo njegovo opterećenje i smanjujemo broj poslatih zahteva.

Dakle, kako je HTTP baziran na zahtev–odgovor protokolu i kao takav ne podržava simultano slanje podataka (full-duplex konekcija), WebSocketi rešavaju upravo ovaj problem.

Prednosti WebSoketa

Pošto se WebSocketi nadograđuju na HTTP, prednosti WebSoketa su:

- konstantna, neprekidna, konekcija između klijenta i servera gde se podaci šalju klijentu bez obzira na to da li je poslao zahtev ili ne;
- komunikacioni tok može teći u oba pravca (klijent–server, server–klijent) u toku konekcije (full-duplex);
- malo kašnjenje – brzina primanja i slanja poruka je veća.

Upotreba WebSoketa

WebSocketi se koriste kada nam je bitna brzina komunikacije. Koriste se najviše za:

- čet aplikacije;
- praćenje rezultata uživo;
- ažuriranje tokova podataka na društvenim mrežama u realnom vremenu;
- multiplejer igre.

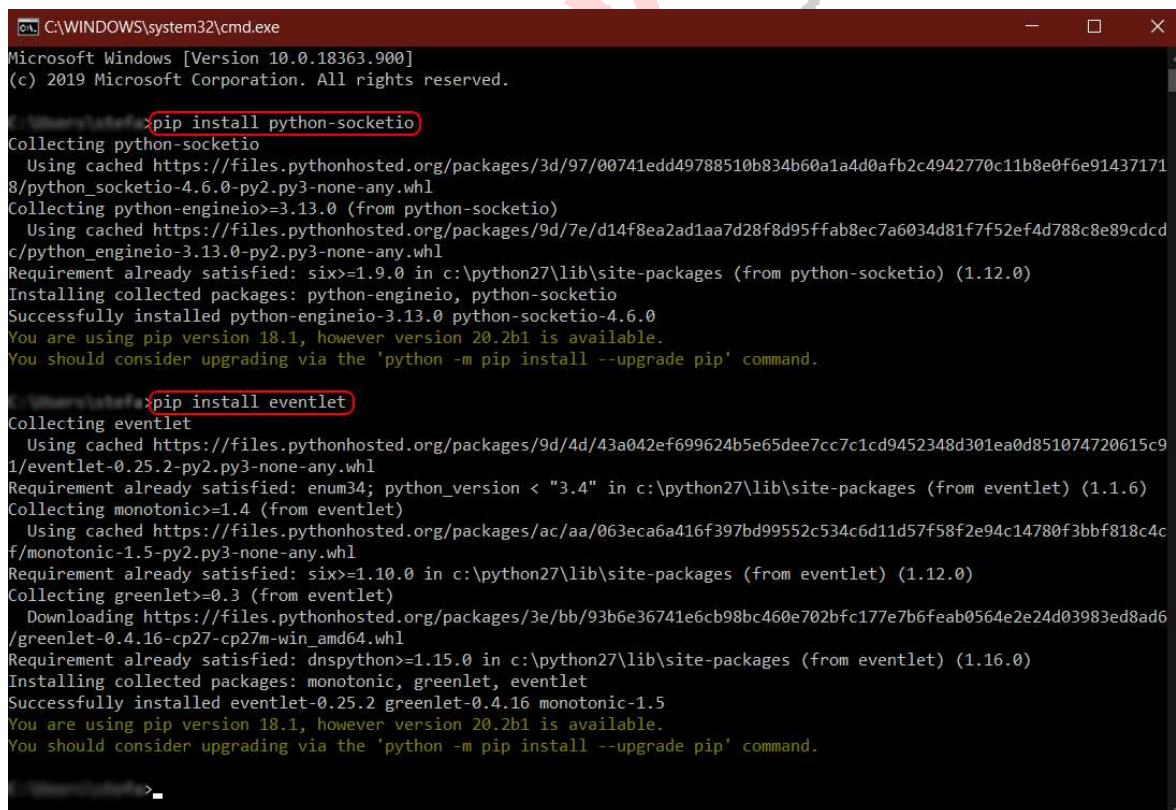
Implementacija WebSocketa u Pythonu

Implementaciju WebSocketa u Pythonu ćemo izvršiti pomoću dve biblioteke koje nisu deo liste ugrađenih biblioteka već se moraju naknadno instalirati. Te biblioteke su:

- `socketio` – biblioteka koja nam omogućava rad sa WebSocketima, bazirana na originalnoj biblioteci koja je napisana u JavaScriptu. Više o njoj možete saznati na zvaničnoj [stranici](#);
- `eventlet` – biblioteka koja pruža olakšano, konkurentno upravljanje mrežom.

Instalacija se vrši korišćenjem pakalnog instalera – PIP-a, koji dolazi kao ugrađeni deo Python instalacije. Pokreće se kroz komandnu liniju kucanjem komande `pip`. Za instalaciju paketa se koristi naredba `install` komande `pip`, nakon koje sledi tačno ime paketa. Važno je napomenuti da imena paketa nekad nisu ista kao imena koja u Python program uvozimo `import` naredbom i da se tačan naziv paketa može videti na njegovoj zvaničnoj stranici. Neretko se dešava da na zvaničnoj stranici paketa postoji i detaljno uputstvo kako instalirati taj paket. Naredbe za instalaciju dve gorepomenute biblioteke su: `pip install eventlet` i `pip install python-socketio` (u zavisnosti od računara ova biblioteka zahteva instalaciju dodatne biblioteke `python-socketio[client]` što možemo učiniti naredbom: `pip install python-socketio[client]`).

Za brisanje instaliranih paketa se koristi naredba `pip uninstall packet-name`.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

> pip install python-socketio
Collecting python-socketio
  Using cached https://files.pythonhosted.org/packages/3d/97/00741edd49788510b834b60a1a4d0afb2c4942770c11b8e0f6e914371718/python_socketio-4.6.0-py2.py3-none-any.whl
Collecting python-engineio>=3.13.0 (from python-socketio)
  Using cached https://files.pythonhosted.org/packages/9d/7e/d14f8ea2ad1aa7d28f8d95ffab8ec7a6034d81f7f52ef4d788c8e89cdcd/c/python_engineio-3.13.0-py2.py3-none-any.whl
Requirement already satisfied: six>=1.9.0 in c:\python27\lib\site-packages (from python-socketio) (1.12.0)
Installing collected packages: python-engineio, python-socketio
Successfully installed python-engineio-3.13.0 python-socketio-4.6.0
You are using pip version 18.1, however version 20.2b1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

> pip install eventlet
Collecting eventlet
  Using cached https://files.pythonhosted.org/packages/9d/4d/43a042ef699624b5e65dee7cc7c1cd9452348d301ea0d851074720615c91/eventlet-0.25.2-py2.py3-none-any.whl
Requirement already satisfied: enum34; python_version < "3.4" in c:\python27\lib\site-packages (from eventlet) (1.1.6)
Collecting monotonic>=1.4 (from eventlet)
  Using cached https://files.pythonhosted.org/packages/ac/aa/063eca6a416f397bd99552c534c6d11d57f58f2e94c14780f3bbf818c4cf/monotonic-1.5-py2.py3-none-any.whl
Requirement already satisfied: six>=1.10.0 in c:\python27\lib\site-packages (from eventlet) (1.12.0)
Collecting greenlet>=0.3 (from eventlet)
  Downloading https://files.pythonhosted.org/packages/3e/bb/93b6e36741e6cb98bc460e702bfc177e7b6feab0564e2e24d03983ed8ad6/greenlet-0.4.16-cp27-cp27m-win_amd64.whl
Requirement already satisfied: dnspython>=1.15.0 in c:\python27\lib\site-packages (from eventlet) (1.16.0)
Installing collected packages: monotonic, greenlet, eventlet
Successfully installed eventlet-0.25.2 greenlet-0.4.16 monotonic-1.5
You are using pip version 18.1, however version 20.2b1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Slika 6.1. Prikaz uspešne instalacije ovih biblioteka

Uz pomoć ove biblioteke ćemo implementirati WebSocket protokol na primeru jednostavne čet aplikacije (server: websockets_local_server.py) koja će moći da primi više klijenata (websockets_local_client.py) i da prikazuje poruke između njih. Klijentska strana će biti tako organizovana da korisnik prvo funkcijom input() upisuje svoje ime, a potom socketio biblioteka preuzima kreiranje konekcije, kao i slanje i primanje poruka. Ovo ime sa klijentske strane nam je bitno jer će nam pomoći u identifikaciji povezanih klijenata na serverskoj strani.

Pitanje

Na kom transportnom protokolu se WebSocketsi zasnivaju?

- UDP
- **TCP**
- TELNET

Objašnjenje:

Tačan odgovor je da se WebSocketsi zasnivaju na TCP protokolu, isto kao i HTTP protokol.

WebSocket klijent

Iako smo pomenuli dve biblioteke koje će nam biti potrebne, za klijent nam je neophodna samo jedna, i to socketio. Ime korisnika koji će se povezati na naš WebSocket server prihvatamo pomoću input() ugrađene funkcije i smeštano u promenljivu username.

WebSocket klijent objekat ćemo instancirati socketio.Client() pozivom i taj objekat smestiti u promenljivu sio_client. Na ovaj način smo samo kreirali klijentski objekat. Povezivanje na sam server ćemo jednostavno učiniti metodom sio_client.connect() kojoj prosleđujemo soket na koji se povezujemo (ip adresa:port).

Napomena

WebSocketsi mogu koristiti i portove rezervisane za HTTP protokol (8080 i 80).

Nakon ovih naredbi, ako je konekcija uspešna, možemo nastaviti sa implementacijom klijenta. Pošto je WebSocket baziran na događajima, moramo definisati funkcije koje će odgovarati tim događajima. Ti događaji mogu biti povezivanje, gašenje konekcije ili slanje poruke (rezervisana imena događaja po socketio biblioteci su connect, disconnect i message). Definisanje funkcija koje odgovaraju ovim osnovnim događajima (a koje se moraju nazvati po njima) činimo tako što te naše funkcije dekoriramo sio_client.event funkcijom. Takođe, kako WebSocket protokol podržava i implementaciju sopstvenih događaja, i za njih koristimo istu sio_client.event metodu kojom ćemo ih dekorisati. Na ovaj način smo „registrovali“ funkciju za dati događaj.

Na primer, ako je konekcija uspešna, izvršiće se događaj connect, što u kontekstu Pythona znači da će se izvršiti funkcija def connect().

Primer dekorisanja jednog osnovnog događaja po imenu connect

```
@sio_client.event

def connect():

    print('connection established')

Primer dekorisanja sopstvenog događaja po imenu server_response:

@sio_client.event

def server_response(data):

    print('this is server response', data)
```

Za naš primer je dovoljno definisati funkcije koje odgovaraju connect i disconnect događaju, kao i zasebnu funkciju send_msg(), koja će primiti poruku koju unese klijent i slati je serveru.

Logika naše čet aplikacije je takva da, odmah po povezivanju, želimo da počnemo sa slanjem poruka ka serveru. Takođe, želimo da omogućimo klijentu da konstantno šalje poruke dokle god ne prekine vezu. Ovo se može implementirati pomoću jedne beskonačne while petlje. Telo ove petlje bi činile:

- input() funkcija – funkcija kojom će korisnik uneti svoju poruku, koju ćemo smestiti u promenljivu msg;
- sio_client.emit() – funkcija kojom šaljemo (emitujemo) događaj sa porukom ka serveru. U ovoj funkciji su nam bitna dva parametra. Prvi od njih, tipa string, predstavlja ime događaja koji šaljemo. Za to ime događaja na serverskoj strani treba da postoji funkcija sa istim takvim imenom koja će rukovati tim događajem. Drugi parametar je tipa rečnik (u našem primeru je tipa rečnik, ali može biti str, bytes, list ili dict); preko njega možemo poslati željene podatke – u ovom slučaju želimo da pri svakom slanju pošaljemo korisnikovu poruku, kao i njegovo ime. Dakle, ako emitujemo događaj po imenu my_message, na serverskoj strani moramo imati funkciju def my_message(): pass.

Na ovaj način smo implementirali WebSocket klijenta pomoću socketio biblioteke. Važno je napomenuti da se u praksi klijent češće sreće kao sam pregledač preko čijeg interfejsa korisnik komunicira sa serverom.

websockets_local_client.py

```
import socketio

username = input("Your username is:")

sio_client = socketio.Client()
```

```

def send_msg():
    while True:
        msg = input("Enter message: ")
        sio_client.emit('my_message', {'msg': msg, 'name': username})
        sio_client.sleep(1) # wait for 1 sec and ask user again for
        message

@sio_client.event
def connect():
    print('connection established')
    send_msg()

sio_client.connect('http://localhost:5000')

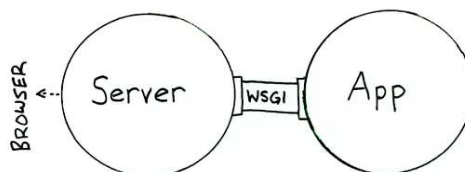
sio_client.wait() # Wait until server ends the connection

```

WebSocket server

Implementacija WebSocket servera se takođe bazira na definisanju funkcija koje će obrađivati željene događaje, a dekorisane su metodama WebSocket serverskog objekta.

Za implementaciju ovog servera uvozimo biblioteke eventlet i socketio. Biblioteka socketio nam omogućava da naredbom `socketio.Server()` kreiramo serverski objekat koji ćemo nazvati `sio_server`. Ono što se razlikuje od klijentske strane je to što ovaj objekat moramo *pustiti u pogon*. Tačnije, potreban nam je način da naš kod servera postavimo na server. Ovo se čini korišćenjem posebnog okruženja (tzv. posredničkog softvera) po imenu WSGI, koji je svojstven Pythonu. WSGI (Web Server Gateway Interface) posreduje kada je reč o zahtevima klijenta ka serveru i omogućava povezivanje korisničkog zahteva ka samom serveru sa odgovarajućom funkcionalnošću u Python skripti ili Python aplikaciji.



Slika 6.2. Jednostavan prikaz funkcionisanja WSGI-a¹

Iz ovih razloga je potrebno da naš serverski objekat prosledimo WSGI okruženju kako bi taj naš serverski objekat mogao da prima klijentske zahteve i događaje. Ovo činimo naredbom: `socketio.WSGIApp(sio_server)` koju smeštamo u promenljivu `app`. Sada, sa tom promenljivom možemo pustiti svoj server u rad.

Pomoću `eventlet` biblioteke stavićemo server u stanje osluškivanja naredbom:

```
eventlet.wsgi.server(eventlet.listen(('localhost', 5000)), app)
```

Treba obratiti pažnju na to da je potrebno koristiti iste brojeve portova i na klijentskoj i na serverskoj strani.

Nakon poslednje naredbe naš server je spreman za nadolazeće zahteve. Ali, pre nego što pokrenemo server, moramo definisati i scenario šta će se dešavati kada stigne zahtev za povezivanje od klijenta, a šta kada dođe događaj `my_message` koji smo definisali na klijentskoj strani.

Na serverskoj strani, definisanje funkcija koje odgovaraju WebSocket događajima koje očekujemo se vrši na sličan način kao i na klijentskoj, s tim što je potrebno da te funkcije definišemo sa minimum jednim parametrom (ako klijent uz slanje događaja ne šalje nikakve podatke) koji prikazuje identifikacioni broj klijenta. Izuzetak od ovog pravila je jedino funkcija `connect`, koja se uvek definiše sa dva parametra, od kojih jedan predstavlja identifikacioni broj klijenta, a drugi predstavlja zaglavlje klijentskog inicijalnog zahteva tipa rečnik. Dakle, kada klijent obavesti WebSocket server da šalje `connect` zahtev, pozvaće se serverska funkcija `connect(sid, headers)`. Kao što na klijentskoj strani dekoriramo funkcije, to radimo i na serverskoj strani, i to `sio_server.event` funkcijom.

Na serverskoj strani ćemo za definisanje funkcije koja će rukovati `my_message` događajem koji će služiti za ispis klijentske poruke definisati istoimenu funkciju `def my_message()`. Argumenti ove funkcije će biti podrazumevani identifikacioni broj klijenta, ali i argument `data` koji je tipa rečnik i koji predstavlja *te/o* čitave poruke koju klijent šalje. Iz klijentskog koda vidimo da je to rečnik sa dva polja: `name`, koje označava ime korisnika, i `uneta` poruka, koja se prosleđuje po ključu `msg`.

websockets_local_server.py

```
import eventlet

import socketio

sio_server = socketio.Server()

app = socketio.WSGIApp(sio_server)

@sio_server.event

def connect(sid, headers):
```

¹ <https://www.appdynamics.com/blog/engineering/an-introduction-to-python-wsgi-servers-part-1/>


```

        print('Client connected: ', sid)

@sio_server.event

def my_message(sid, data):

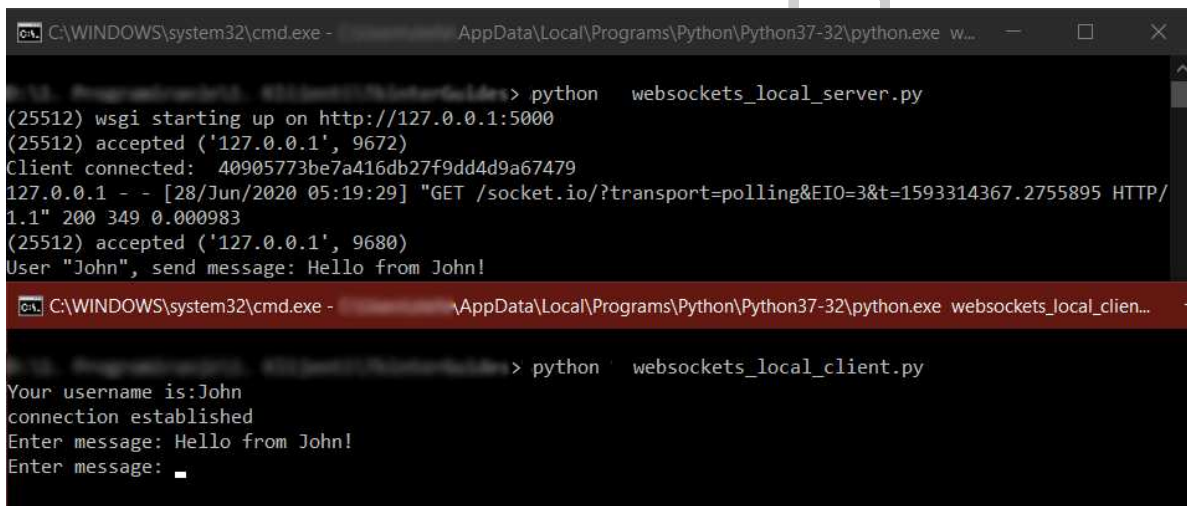
    print('User "{}", send message: {}'.format(data['name'], data['msg']))

eventlet.wsgi.server(eventlet.listen(('localhost', 5000)), app)

```

Pokretanje i analiza rezultata

Za testiranje klijenta i servera potrebno je pokrenuti dva različita komandna prozora i pozicionirati se u direktorijumu gde se fajlovi `websockets_local_client.py` i `websockets_local_server.py` nalaze. Prvo se pokrene serverska strana, a potom klijentska.



```

C:\WINDOWS\system32\cmd.exe - AppData\Local\Programs\Python\Python37-32\python.exe w...
python websockets_local_server.py
(25512) wsgi starting up on http://127.0.0.1:5000
(25512) accepted ('127.0.0.1', 9672)
Client connected: 40905773be7a416db27f9dd4d9a67479
127.0.0.1 - - [28/Jun/2020 05:19:29] "GET /socket.io/?transport=polling&EIO=3&t=1593314367.2755895 HTTP/1.1" 200 349 0.000983
(25512) accepted ('127.0.0.1', 9680)
User "John", send message: Hello from John!

C:\WINDOWS\system32\cmd.exe - \AppData\Local\Programs\Python\Python37-32\python.exe websockets_local_clien...
python websockets_local_client.py
Your username is:John
connection established
Enter message: Hello from John!
Enter message:

```

Slika 6.3. Prikaz pokretanja klijenta i servera kroz komandnu liniju

Prvo što primetimo kada pokrenemo serversku stranu jeste da se zapravo ništa ne dešava osim što server osluškuje nadolazeće zahteve na zadatom soku (ispis (25512) *wsgi starting up on http://127.0.0.1:5000*). Prilikom pokretanja klijentskog fajla, prvo upisujemo ime kojim želimo da nas server identifikuje i nakon toga nastupa povezivanje klijenta i servera. Kada dođe do ostvarivanja konekcije, server ispisuje (25512) *accepted ('127.0.0.1', 9672)*, što znači da je u tom trenutku klijentskoj konekciji dodelio novi soket (ostvorena HTTP konekcija). Dalje, klijent šalje zahtev o promeni na WebSocket protokol i u tom trenutku server ispisuje liniju:

```

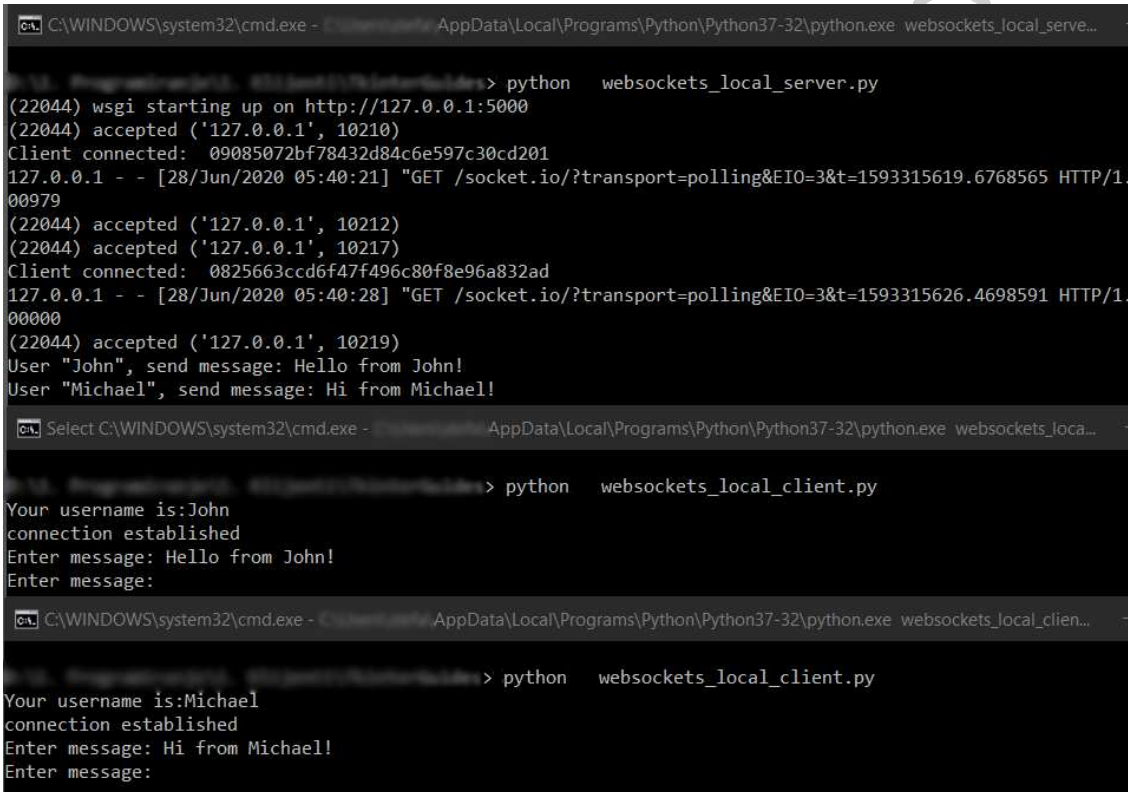
127.0.0.1 - - [28/Jun/2020 05:19:29] "GET
/socket.io/?transport=polling&EIO=3&t=1593314367.2755895 HTTP/1.1" 200
349 0.000983

```


kojom prikazuje taj, klijentski, GET zahtev, posle koga sledi poruka o prihvatanju tog zahteva: (25512) accepted ('127.0.0.1', 9680). Kako je zahtev prihvaćen, tako se na klijentskoj strani ispisuje poruka *connection established* i slanje može početi.

U ovom trenutku klijent može početi sa slanjem poruka, a server je u stanju čekanja i osluškuje događaj `my_message`, preko kojeg dolazi klijentska poruka.

Takođe je moguće povezati dva i više klijenata na naš server (svaki u posebnom prozoru komandne linije). Jedan takav primer izgleda ovako:



```
C:\WINDOWS\system32\cmd.exe - AppData\Local\Programs\Python\Python37-32\python.exe websockets_local_serve...
C:\ProgramData\Python\Python37-32\python.exe > python websockets_local_server.py
(22044) wsgi starting up on http://127.0.0.1:5000
(22044) accepted ('127.0.0.1', 10210)
Client connected: 09085072bf78432d84c6e597c30cd201
127.0.0.1 - - [28/Jun/2020 05:40:21] "GET /socket.io/?transport=polling&EIO=3&t=1593315619.6768565 HTTP/1.1" 200 00979
(22044) accepted ('127.0.0.1', 10212)
(22044) accepted ('127.0.0.1', 10217)
Client connected: 0825663ccd6f47f496c80f8e96a832ad
127.0.0.1 - - [28/Jun/2020 05:40:28] "GET /socket.io/?transport=polling&EIO=3&t=1593315626.4698591 HTTP/1.1" 200 00000
(22044) accepted ('127.0.0.1', 10219)
User "John", send message: Hello from John!
User "Michael", send message: Hi from Michael!

C:\WINDOWS\system32\cmd.exe - AppData\Local\Programs\Python\Python37-32\python.exe websockets_local_...
C:\ProgramData\Python\Python37-32\python.exe > python websockets_local_client.py
Your username is:John
connection established
Enter message: Hello from John!
Enter message:

C:\WINDOWS\system32\cmd.exe - AppData\Local\Programs\Python\Python37-32\python.exe websockets_local_clie...
C:\ProgramData\Python\Python37-32\python.exe > python websockets_local_client.py
Your username is:Michael
connection established
Enter message: Hi from Michael!
Enter message:
```

Slika 6.4. Prikaz pokretanja servera i dva klijenta kroz komandnu liniju

U ovoj nastavnoj jedinici smo realizovali WebSocket protokol na osnovu jednostavne čet aplikacije. Međutim, WebSocketi predstavljaju moćnu tehnologiju koja se može koristiti i za mnogo ozbiljnije web aplikacije.

Rezime

- Anketiranje servera (server polling) je metoda kojom klijent konstantno pita server da li je pristigla informacija za njega.
- WebSocketi nam omogućavaju ostvarivanje jedne TCP konekcije između klijenta i servera, koja omogućava dvostranu komunikaciju (full-duplex).

- Zahtev za WebSocket konekciju se šalje serveru sa klijentske strane kroz proces poznat kao WebSocket usaglašavanje.
- Zahtev za usaglašavanje počinje kao običan HTTP zahtev ka serveru i sadrži i Upgrade polje zaglavlja, koje nagoveštava serveru kako klijent pokušava da napravi WebSocket konekciju.
- Biblioteka socketio, koja nam omogućava rad sa WebSocketima, bazirana je na originalnoj biblioteci koja je napisana u JavaScriptu.
- WebSocket klijent objekat ćemo instancirati kao socketio.Client().
- WebSocket server objekat ćemo instancirati kao socketio.Server().
- Za slanje događaja i poruka sa njima koristimo metodu emit().
- Za svaki događaj koji želimo da koristimo, bez obzira na to da li je reč o klijentskoj ili serverskoj strani, moramo definisati i odgovarajuću funkciju i dekorisati je.

